Modeling Concurrent Behavior through Consistent Statechart Views

Gregor Engels[1], Luuk Groenewegen[2], and Jochen M. Küster[1]

[1] University of Paderborn, Dept. of Computer Science, D-33095 Paderborn, Germany
email: {engels, jkuester}@upb.de, phone: +49 − 5251 − 603357
[2] Leiden University, LIACS, P.O. Box 9512, NL-2300 RA Leiden, The Netherlands
email: luuk@liacs.nl, phone: +31 − 71 − 5277139

1 Introduction

Concurrent systems typically consist of multiple self-contained components that communicate with each other by exchanging messages over connectors. Modeling the behavior of such systems is difficult due to the concurrent execution of components and has been dominated by formal approaches such as CSP [3], CCS [5] and Petri nets. Describing communication in terms of discrete mathematical structures these approaches allow the proving of desired properties. However, they focus on the communication aspect neglecting the modeling of object structures and are still not widely applied by software engineers due to their formality.

Object-oriented modeling [7] combines structural and behavioral modeling of the system using structural, dynamic and functional models. Structural models allow the modeling of static aspects of the system. Dynamic models focus on behavioral aspects of the system. Functional models describe the effect of operations. Nowadays, the Unified Modeling Language (UML) [6] is the accepted industrial standard for object-oriented modeling.

Modeling concurrent systems with UML raises the issue of how concurrent behavior can be properly modeled using existing techniques of UML. Since the advent of the UML, several extensions have been proposed in order to enable the modeling of problems of a specific domain. The UML-RT [8] is an extension of the UML for modeling complex real-time systems and it is in the process of becoming a UML profile. It incorporates the special notions of an active object called capsule and connectors connecting capsules. As a consequence, UML-RT seems to be the candidate for modeling concurrent systems consisting of components (or capsules) intertwined by connectors as a real-time system can be considered to be a special form of a concurrent systems.

In addition to the concepts of capsules and connectors, UML-RT introduces the concepts of ports, protocol roles and protocols. A port is a connection point between a capsule and a connector. A port is associated with a protocol role defining the signals received and sent via the port to the port of a capsule at the other end of the connector. Several protocol roles together form a protocol which is in terms of the UML a specialization of a collaboration.

Statecharts [2] are used in UML-RT for specifying the behavior of capsules. A capsule statechart describes how the capsule reacts to signals received via one of its ports by calling operations of the capsule. Optionally, a protocol can also be associated with a statechart specifying the allowed interaction, i. e. the ordering of messages exchanged by the protocol participants. However, the actual performing of an interaction is incorporated in the statecharts of the capsules that take part in the protocol. This results into two descriptions of the interaction, one given by the protocol statechart and the other one modeled implicitly in the capsule statecharts. As a consequence, the consistency between these two descriptions must be ensured.

Consistency within the model is important for the development process and a clear consistency concept is still missing [1]. A consistency concept must define what consistency means and how to establish consistency within a model. We can distinguish between consistency on a syntactical and semantical level. In our particular case, statecharts of two capsules taking part in a common protocol must be syntactically consistent in so far that messages issued by one capsule must be understandable by the capsule it is connected to. From a semantic point of view, the order of messages resulting from the execution of capsule statecharts must conform to the order specified in the protocol statechart.

In our opinion there is a need for describing interactions between capsules such that each interaction is modeled explicitly as well as consistently. Currently, elements of concurrent behavior are distributed to different capsule statecharts, leaving the interaction only implicitly modeled. This results into the difficulty of checking the consistency with the explicit description of the interaction given by a protocol statechart. As there is also no concept of consistency for these two descriptions we deduce that neither UML nor its extension UML-RT support the modeling of concurrent behavior adequately. We are therefore interested in answering the question how consistency can be achieved in this particular case. Thereby, we aim at overcoming the lack of verification properties compared to formal approaches.

In the remainder of our position statement we first elaborate on the applications of statecharts in UML-RT for modeling concurrent behavior by presenting a small example. Then we indicate our approach towards overcoming the consistency problem of statecharts in UML-RT.

2 Problem Description

In order to illustrate the consistency problem we introduce a small example consisting of two capsules connected to each other by a connector. Figure 1 shows the so-called capsule collaboration diagram view of the example, enriched with the statecharts used by UML-RT.

The two capsules (named *CapsuleA* and *CapsuleB*) both have one port (named $P1$ and $P2$ respectively) by which they are connected to the other one via a connector. The ports are associated to *protocol roles* (named *RoleA* and *RoleB*) specifying the signals sent and received via the port (not mentioned in

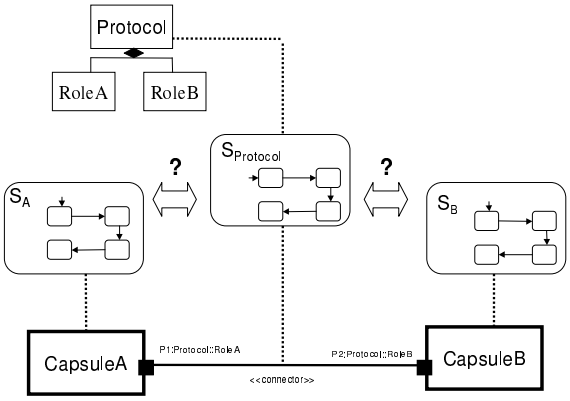

Fig. 1. Example in UML-RT

this diagram). Two or more protocol roles form a protocol, in our case *RoleA* and *RoleB* form the protocol *Protocol*.

From the point of view of behavior modeling each capsule is associated to a statechart specifying states and state transitions of the capsule. In our example, statecharts S_A and S_B specify the behavior of *CapsuleA* and *CapsuleB*, respectively. Capsule statecharts describe how capsules react to signals received via its ports and when signals are sent via its ports. State transitions of capsule statecharts may also include the calling of capsule operations. Note that capsule operations cannot be called from outside capsules but only from the capsule itself. A protocol statechart may optionally be associated to a protocol describing the valid order of messages exchanged within the protocol. In our case, $S_{Protocol}$ is the statechart for the protocol.

For our example, we can formulate the above mentioned questions of consistency more precisely. From a syntactical point of view, each statechart must be compatible with the protocol roles associated to the ports of the capsule. For example, a signal sent over port $P1$ by S_A must occur in the signal compartment of *RoleA*. Furthermore, roles of a protocol must be compatible with each other meaning that signals specified as outgoing signals in one role must be specified as incoming signals in the other role and vice versa.

With respect to semantic consistency, statecharts S_A and S_B must be compatible with each other, meaning that a signal sent over the connector from *CapsuleA* to *CapsuleB* must be understood by S_B. This means that there must be a suitable transition in S_B which is triggered by the signal. Additionally, the order of messages exchanged via the connector resulting from the execution of *CapsuleA* and *CapsuleB* must be consistent to the order specified in the protocol statechart $S_{Protocol}$.

Consistency between S_A and S_B is important for a correct execution of the system. Consistency of the order of messages with the specification in the protocol statechart $S_{Protocol}$ is necessary for the following reasons. Given a protocol

statechart, independent developers might be forced to ensure that their capsule conforms to the given protocol statechart. Conversely, having developed a system one might want to extract the protocol statechart for being able to exchange one capsule. A smooth integration of a new capsule can only be ensured if it conforms to the protocol statechart.

This gives rise to the question how the different statecharts are related. An answer to this question has to define a notion of consistency between the various statecharts such that their mutual influencing can be formulated, indicated and analysed.

3 Our approach

Based on the above observations our approach is as follows. We are going to introduce two different notions of a protocol. A *protocol template* describes all possible orders of messages exchanged between objects and is of a descriptive nature being possibly non-deterministic. A protocol template is specified using a protocol statechart. State transitions between protocol template states are labelled with a signal name and an (abstract) capsule name sending that signal.

A *protocol instance* can be viewed as one particular implementation of a protocol template. A protocol instance is the result of the interaction of two or more capsules participating in the protocol. In our approach, capsule behavior is specified by capsule statecharts. The protocol instance therefore arises from the statecharts of the capsules participating in the protocol. But there is more: in principle, connector behavior can be rather different, e. g. the connector can be order-preserving or it can have a limited or unlimited capacity. As the actual connector behavior is relevant for the protocol instance, the statechart of the protocol instance must additionally take into account the behavior of the connector. In our opinion, connector behavior can be specified using an additional statechart for modeling the dynamic behavior of a connector.

In Figure 2 our approach is illustrated. In order to construct the statechart of the protocol instance from the statecharts of the capsules the notion of a *statechart view* will be introduced which specifies the part of the statechart of the capsule relevant to some particular interaction. The statechart of the capsule specifies how signals are sent over various connectors. For the consistency with the protocol template statechart, one is only interested in the signals sent over the connector the protocol is associated to. As a consequence, suitable reduction algorithms for statecharts are needed which enable the construction of a statechart view from a capsule statechart. Intuitively, a statechart view can be computed by taking into account only transitions involving the sending or receiving of signals over the port in focus. In our example, $S_{AViewPortP1}$ and $S_{BViewPortP2}$ are the statechart views computed from both capsule statecharts.

For the actual interaction between the two capsules it is not sufficient to concentrate on the statechart views alone because, as we already pointed out, the behavior of the underlying connector is also important. We therefore introduce a connector statechart specifying the dynamic behavior of a connector. A

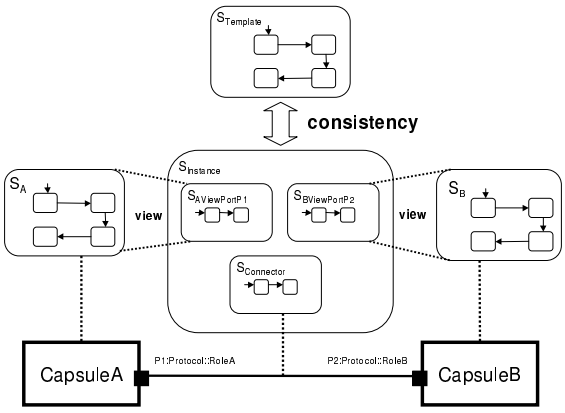

Fig. 2. Statecharts in our approach

connector statechart consists of two substatecharts, each specifying the behavior of the connector in one communication direction.

On the basis of the connector statechart and all statechart views the protocol instance can be constructed by computing the product of the statechart views taking into account restrictions imposed by the connector.

Having established the concept of protocol instance and template we are going to define the consistency of a protocol instance with a protocol template. The protocol instance statechart must conform to the protocol template statechart. With respect to theory of statecharts, this problem raises the question of statechart equivalence [4]. In practice, there remains the question of defining suitable algorithms and develop tool support.

Summarizing our position is as follows: We think that modeling of concurrent behavior is important for the development of concurrent systems and for the modeling of concurrency within object-oriented systems. Modeling concurrent behavior should be explicit and needs a clear concept of consistency which is currently lacking in UML. Our approach distinguishes between a protocol template describing all possible interactions and a protocol instance describing a particular implementation of a protocol template. A clear consistency concept between the protocol template and the protocol instance should be defined. By computing statechart views from each capsule taking part in a protocol and the introduction of an additional connector statechart we aim at computing the protocol instance statechart and thereby establishing consistency in concurrent behavior.

References

1. G. Engels and L. Groenewegen. Object-oriented modeling: A roadmap. In Anthony Finkelstein, editor, *Future Of Software Engineering 2000*, pages 105–116. ACM,

June 2000.
2. D. Harel. Statecharts: A visual formulation for complex systems. *Science of Computer Programming*, 8(3):231–274, June 1987.
3. C. A. R. Hoare. *Communcating Sequential Processes.* Prentice Hall, 1985.
4. A. Maggiolo-Schettini, A. Peron, and S. Tini. Equivalences of statecharts. In Ugo Montanari and Vladimiro Sassone, editors, *CONCUR '96: Concurrency Theory, 7th International Conference*, volume 1119 of *Lecture Notes in Computer Science*, pages 687–702, Pisa, Italy, 26–29 August 1996. Springer-Verlag.
5. R. Milner. *Communication and Concurrency.* International Series in Computer Science. Prentice Hall, 1989. SU Fisher Research 511/24.
6. Object Modeling Group. *Unified Modelling Language Specification, version 1.3*, June 1999. URL: `uml.shl.com:80/docs/UML1.3/99-06-08.pdf`.
7. J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, and W. Lorensen. *Object-Oriented Modeling and Design.* Prentice Hall, Englewood Cliffs, New Jersey, USA, 1991.
8. B. Selic. Using UML for modeling complex real-time systems. *Lecture Notes in Computer Science*, 1474:250–262, 1998.